## REMARKS

Claims 1-19 were pending. With this Response, claims 1-4, 6-9, 11-14, and 16-19 are amended. Claims 5 and 10 are cancelled. Therefore, claims 1-4, 6-9, 11-19 are pending.

## CLAIM OBJECTIONS

Claims 16-19 were objected to because of informalities and have been appropriately amended.

## REJECTIONS UNDER 35 U.S.C. § 112

Claims 2, 5, 6, 10 and 14 were rejected under 35 U.S.C. § 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the invention. More particularly, the Office Action at page 4 notes what are considered to be inconsistencies in the claim language.

Applicants respectfully submit that the amendments to the claims overcome the objection and thus, respectfully request the Examiner to withdraw the rejection to the claim under 35 U.S.C. §112.

## REJECTIONS UNDER 35 U.S.C. § 101

The Office Action rejected claims 9-12 and 16-19 under 35 U.S.C. § 101 as being directed toward "non-statutory subject matter." More particularly, at page 5, item 12, the Office Action rejects claims 9-12 alleging that "such claimed software module/organizer is software program listings per se and it does not define any structural and functional interrelationships between the computer program and other claimed elements of a computer, which permit the computer program's functionality to be realized." This analysis is incorrect as software programs inherently have functional interrelationships between the computer program and other structural elements of a computer. Examiner states, at page 5, last paragraph, "these rejections can be overcome by adding computer hardware components e.g., memory, and processor into the claims that permit the computer program's functionality to be realized."

Applicants have introduced the clarifying limitation, "processor" into the amended claims, so as to comport with the recommendation set forth in the Office Action. Applicants have also clarified the claim language to recite a "detection of a portion of the computer program

outside of the critical section that could be executed by the processor," so as to eliminate any potential for ambiguity or erroneous interpretation of Applicants' claimed embodiment and making expressly clear that Applicants' claimed embodiment is directed toward patentable subject matter under 35 U.S.C. § 101.

Furthermore, at page 6, the Office Action rejected claims 16-19 under 35 U.S.C. § 101 as being directed toward "non-statutory subject matter." In particular, the Office Action states that claim 16 refers to a "transmission over the internet," however, the Office Action alleges "the transmission medium e.g. signal or wave is only a form of energy that is not a tangible physical article or object and it does not fall within either of the two definitions of manufacture. Claims 17-19 are rejected because they depend from claim 16 and contain the same deficiency.

This analysis is incorrect. In accordance with established case law under in view of *In re Nuijten*, 500 F.3d 1346 (Fed. Cir. 2007), propagated signals or energy waves are incapable of storing, and thus are not patentable subject matter. Accordingly, a claim which recites a "computer readable storage medium," such as that which Applicants recite in independent claim 16, is directed to patentable subject matter under *In re Nuijten*, because the court's opinion in that case holds that "signals," or "transitory" embodiments, cannot "store" anything, thus rendering the Office's interpretation of Applicants' claimed "machine-readable storage medium" as being capable of somehow covering "non-statutory transitory media such as signals or transmission media" erroneous.

Nevertheless, in the interests of advancing prosecution toward as expeditious of an allowance as feasible, Applicants have introduced the clarifying limitation, "non-transitory" prior to the term, "computer readable storage medium," so as to make express that Applicants' claimed embodiment is directed toward patentable subject matter under 35 U.S.C. § 101.

Accordingly, Applicants submit that the amendments to the claims overcome the present rejection and thus, respectfully request the Examiner to withdraw the rejection to the claim under 35 U.S.C. §101.

## REJECTIONS UNDER 35 U.S.C. § 103

Claims 1-19 were rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent Application Publication No. 2005/0108695A1 of Li et al. (hereinafter "Li") in view of

U.S. Patent No. 5,712,791 of Lauterbach (hereinafter "Lauterbach "). Applicants respectfully disagree. In particular, independent claim 1, as amended herein, recites:

> A computer implemented method for rearranging a computer program comprising:
> organizing the computer program **logically** into a plurality of blocks;
> **constructing a dependency graph based on the organization of the plurality of blocks in the computer program**;
> **determining a critical section** included in the dependency graph;
> **detecting** a portion of the plurality of blocks in the computer program that could be executed outside of the critical section;
> **inserting a plurality of dependency relationships based on the dependency graph between the plurality of blocks** to cause execution of the detected portion of the plurality of blocks in the computer program outside of the critical section; and
> **rearranging** the detected portion of the plurality of blocks to outside the critical section that were inside the critical section based on the inserted plurality of dependency relationships.

Thus, Applicants now expressly recite within independent claim 1 that the method involves "organizing the computer program **logically** into a plurality of blocks, and **constructing a dependency graph** based on the organization of the plurality of blocks in the computer program."

Moreover, Applicants have further amended the claim to expressly recite "**determining a critical section** included in the dependency graph, **detecting** a portion of the plurality of blocks in the computer program that could be executed outside of the critical section, **inserting** a plurality of dependency relationships based on the dependency graphs between the plurality of blocks to **cause execution of the detected portion** of the plurality of blocks in the computer program outside of the critical section, and rearranging the detected portion of the plurality of

blocks to outside the critical section blocks that were inside the critical section based on the inserted plurality of dependency relationships."

Applicants teach within Applicants' originally filed specification that "one embodiment of the invention organizes the computer program into a **plurality of blocks**... and inserts a plurality of dependency relationships between the plurality of blocks..." For example, refer to paragraph 13, which teaches:

> [13] One embodiment of the invention organizes the computer program into a **plurality of blocks**, determines a critical section of the computer program, constructs a dependency graph, recognizes a portion of the computer program that could be executed outside of the critical section, and inserts a **plurality of dependency relationships between the plurality of blocks** to cause execution of the recognized portion of the computer program outside of the critical section.

Thus, inserting "a plurality of dependency relationships between the plurality of blocks" cause the execution outside the critical section of programming instructions that are in the critical section but that could be executed outside the critical section based on the dependency graph.

I. Li fails to disclose claimed limitations:

In the present Office Action at page 7, the Office Action alleges that Li discloses "organizing the computer program into a plurality of blocks (see Fig.2A-2C, Fig.5, method 500 and Paragraphs [0039] [0045]); determining a critical section of the computer program (see Fig. 5, step 510); constructing a control flow graph based on the organization of the computer program (see Fig.5, step 502 and Paragraph [0045]); recognizing a portion of the computer program that could be executed outside of the critical section (see paragraph [0046]); and inserting a plurality of dependency relationships between the plurality of blocks to cause

execution of the recognized portion of the computer program outside of the critical section (see paragraph [0046])."

In support of its assertion, the Office Action makes particular references to Li Figures 2A-2C and Figure 5. Turning to Li specifically, "Figs. 2A-2C depict transformation of a sequential packet processing stage (PPS) into two application program threads, in accordance with one embodiment of the invention." And Fig. 5 is a block diagram illustrating a method for thread partitioning a loop body of a sequential application program, in accordance with one embodiment of the invention.

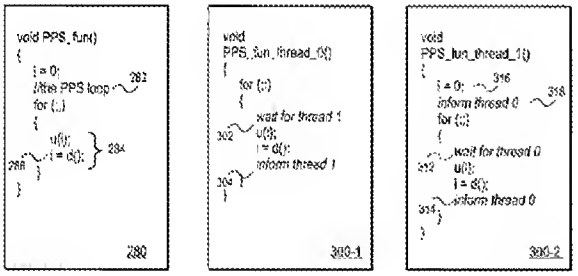


FIG. 2A FIG. 2B FIG. 2C

[0007] FIGS. 2A-2C depict transformation of a sequential packet processing stage (PPS) into two application program threads, in accordance with one embodiment of the invention.

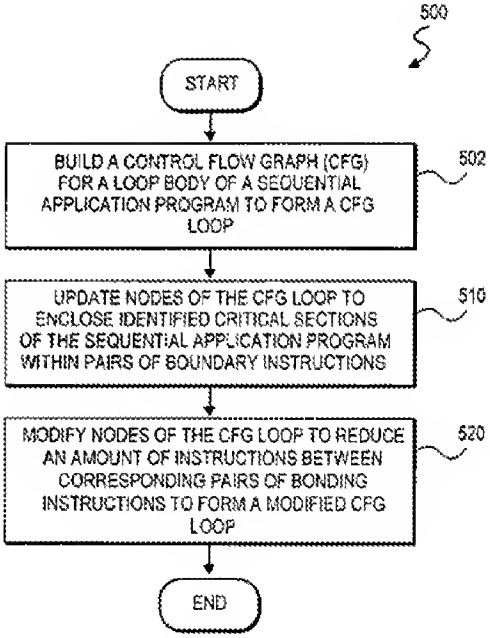


FIG. 5

[0010] FIG. 5 is a block diagram illustrating a method for thread partitioning a loop body of a sequential application program, in accordance with one embodiment of the invention.

Paragraphs 39, 45 and 46 of Li state the following:

> [**0039**] Fig. 5 is a flow chart illustrating a method 500 for thread-partitioning a sequential application program, in accordance with one embodiment of the invention. At process block 502, a

control flow graph (CFG) is built for a loop body of a sequential application program to form a CFG loop. As described herein, a CFG is a graph representing the flow of control of the program, where each vertex represents a basic block, and each edge shows the potential flow of control between basic blocks. A control flow graph has a unique source node (entry). In one embodiment, the formation of the CFG loop, as illustrated with reference to FIGS. 6A and 6B.

[**0045**] As described herein, dataflow analysis is not limited to simply computing definitions and uses of variables (dataflow). Dataflow analysis provides a technique for computing facts about paths through programs or procedures. A prerequisite to the concept of dataflow analysis is the control flow graph (CFG) or simply a flow graph, for example as illustrated with reference to FIG. 6A. A CFG is a directed graph that captures control flow and a part of a program. For example, a CFG may represent a procedure sized program fragment. As described herein, CFG nodes are basic blocks (sections of code always executed in order) and the edges represent possible flow of control between basic blocks. For example, as illustrated with reference to FIG. 6A, control flow graph 600 is comprised of nodes 602-606, as well as edges 603, 605 and back-edge 608.

[**0046**] As described herein, code motion is a technique for inter-block and intra-block instruction reordering (hoisting/sinking). In one embodiment, code motion moves irrelevant code out of identified critical sections in order to minimize the amount of instructions/operations contained therein. To perform the inter-block and intra-block instruction reordering, code motion initially identifies motion candidate instructions. In one embodiment, motion candidate instructions are identified using dataflow analysis. Representatively, a series of dataflow problems are solved to carry out both hoisting and sinking of identified motion candidate instructions.

Thus, Li discloses that it seeks to "build a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop… a CFG is a graph representing the flow of control of the program, where each vertex represents a basic block, and each edge shows the potential flow of control between basic blocks." However, Li expressly and unambiguously

states that the control flow graphs are used to form CFG loops and are not based on "**inserting a plurality of dependency relationships based on the dependency graph between the plurality of blocks**" as claimed by Applicants. For example, refer to Li paragraph 39, which expressly states: "At process block 502, a control flow graph (CFG) is built for a loop body of a sequential application program to form a CFG loop." The CFG disclosed by Li is not implemented using a plurality of blocks, nor is CFG an equivalent to dependency graphs as Applicants recite. At page 8, last paragraph, the Office Action points out that Li fails to disclose "constructing the dependency graph based on the organization of the computer program," and asserts that Lauterbach cures the deficiencies of Li. However, Lauterbach does not disclose and is not offered to disclose the "**inserting a plurality of dependency relationships based on the dependency graph between the plurality of blocks**" limitation as claimed by Applicants. Therefore, Li or Lauterbach, whether considered individually or in combination, fail to disclose this "plurality of dependency relationships based on the dependency graph between the plurality of blocks" limitation and so the claim is non-obvious under 35 U.S.C. § 103 in view of such.

II. Lauterbach fails to disclose claimed limitations:

As noted above, the Office Action at page 8, last paragraph asserts Lauterbach discloses "generating dependency graph based on instructions." Applicants respectfully disagree. In particular, independent claim 1 as amended herein recites:

> A computer implemented method for rearranging a computer program comprising:
>
> organizing the computer program **logically** into a plurality of blocks;
>
> **constructing a dependency graph based on the organization of the plurality of blocks in the computer program**;
>
> **determining a critical section** included in the dependency graph;

> **detecting** a portion of the plurality of blocks in the computer program that could be executed outside of the critical section;
>
> **inserting a plurality of dependency relationships based on the dependency graph between the plurality of blocks** to cause execution of the detected portion of the plurality of blocks in the computer program outside of the critical section; and
>
> **rearranging** the detected portion of the plurality of blocks to outside the critical section that were inside the critical section based on the inserted plurality of dependency relationships.

Applicants teach within Applicants' originally filed specification that generating a dependency graph explicitly requires both **"organizing the computer program logically into a plurality of blocks**," and also explicitly requires **"constructing a dependency graph based on the organization of the plurality of blocks in the computer program."** Lauterbach does not disclose this level of specificity in regards to its dependency graph, and thus, Lauterbach does not disclose these limitations.

In support of its rejection, the Office Action makes particular reference to Lauterbach at Column 3, lines 26-27, stating, that Lauterbach discloses "...**the dependency graph generator generates a dependency graph for a set of program instructions**"; Fig. 2, step 50, "Build Dependency Graph for Trace of instructions" and related text; also see Fig. 3 illustrates a dynamic dependency graph associated with a set of program instructions). No further analysis is provided as to how this conclusion is derived. Nevertheless, turning to Lauterbach specifically, Column 3, lines 23-27 of Lauterbach which is relied upon by the Office Action discloses:

> [Column 3, Paragraph 2] **Memory 28 stores a dependency graph generator 30.** As its name implies, the **dependency graph generator generates a dependency graph for a set of program instructions**. The dependency graph 32 is then stored in the memory 28. Typically, the dependency graph 32 is stored as a data structure with pointers between related instructions.

Thus, Lauterbach discloses a memory that stores a dependency graph generator. Applicants expressly recite in claim 1, "constructing a dependency graph based on the organization of the plurality of blocks in the computer program," where as Lauterbach discloses "dependency graph generator generates dependency graph for a set of program instructions" stored in the memory. As can be appreciated, Lauterbach's dependency graph generator is not the same as "constructing the dependency graph based on the organization of the plurality of blocks in the computer program," as Applicants recite.

Applicants expressly recite that both the "organizing the computer program logically into a plurality of blocks" and also that the "constructing the dependency graph based on the organization of the plurality of blocks in the computer program" is specified *to* the multi-threaded programming instructions, where as Lauterbach discloses only "dependency graph generator." How are the dependency graph "constructed based on the logical organization of the plurality of blocks of the computer program" Lauterbach does not say, and because Lauterbach fails to state what the "organizing" is, Lauterbach does not disclose the limitations set forth by the Applicant.

Because the combination of Li and Lauterbach fail to disclose at least one limitation as Applicants recite in independent claim 1, as amended herein, Applicants respectfully submit that independent claim 1 is patentable over the references and in condition for allowance. Applicants further submit that independent claims 9, 13 and 16, which recite similar limitations, as well as those claims which depend directly or indirectly upon independent claims 1, 9, 13 and 16, and thus incorporate the limitations of their respective parent claims, are also patentable over the references and in condition for allowance for at least the same reasons as stated above with respect to independent claim 1 rejected under 35 U.S.C. § 103.

Accordingly, Applicants respectfully request the Examiner to withdraw the rejection to the claims under 35 U.S.C. §103.

## CONCLUSION

For at least the foregoing reasons, Applicant submits that the rejections have been overcome. Therefore, all pending claims are in condition for allowance, and such action is earnestly solicited. The Examiner is respectfully requested to contact the undersigned by telephone if such contact would further the examination of the present application.

Please charge any shortages and credit any overcharges to our Deposit Account number 02-2666.

Respectfully submitted,
**BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP**

Date: November 15, 2010

/Gregory D. Caldwell/
Gregory D. Caldwell
Reg. No. 39,926
Attorney for Applicant

1279 Oakmead Parkway
Sunnyvale, CA 94085-4040

(503) 439-8778

I hereby certify that this correspondence is being submitted electronically via EFS Web on the date shown below.

Date: November 15, 2010 /Natasha French/
Natasha French